A Binary Translation Framework for Automated Hardware Generation

1st Nuno Paulino
INESC TEC and
Faculty of Engineering of the University of Porto
Porto, PORTUGAL
nuno.m.paulino@inesctec.pt

2nd João Canas Ferreira
INESC TEC and
Faculty of Engineering of the University of Porto
Porto, PORTUGAL
jcf@fe.up.pt

***Index Terms*—accelerator, instruction traces, binary acceleration, HW/SW partitioning, heterogeneous systems**

Hardware specialization is an efficient solution for maximization of performance and minimization of energy consumption. With the emergence of edge systems, designing energy efficient systems becomes increasingly important.

Emerging compilation flows provide automated generation of hardware from high-level languages, i.e., *High-Level Synthesis* (HLS) [Xil17]. Although increasingly appealing, some hardware related knowledge is still required, and thought must be put into the design of the underlying heterogeneous system. Consequently, the effort of hardware/software partitioning and subsequent hardware generation and validation is only worthwhile for cases with well identifiable critical kernels.

In contrast, this work is based on automated detection of workload by analysis of a compiled application, and on the automated generation of specialized hardware modules. By offloading the hardware generation effort to a late stage, developer intervention and effort can be eliminated, and modest but ubiquitous acceleration can be provided transparently. Previous work showns the viability of this approach [PFC17], and this demonstration focuses on on-going work in additional binary analysis and hardware generation capabilities.

We will present the current version of the binary analysis and translation framework. Currently, our implementation is capable of processing ARMv8 and MicroBlaze (32-bit) *Executable and Linking Format* (ELF) files or instruction traces. In the former case, the contents of the compiled program are inspected by resorting to the architecture-specific variants of the *objdump* utility, and in the later case, execution traces are obtained via QEMU [Bel05] emulation.

The framework can interpret the instructions for these two *Instruction Set Architecture*s (ISAs), and decompose their bitfields in order to extract the specific operation and operands. This information is used to detect different types of instruction patterns, which we refer to as *binary segments*. Currently, we can detect four types of segment: frequently occuring short sequences of instructions and frequently occuring basic blocks in the static code, and their counterparts as their occur dynamically in an instruction stream.

This work was supported by the PEPCC project, "PTDC/EEI-HAC/30848/2017", financed by FCT (Fundação para a Ciência e Tecnologia - Portuguese Fundation for Science and Technology).

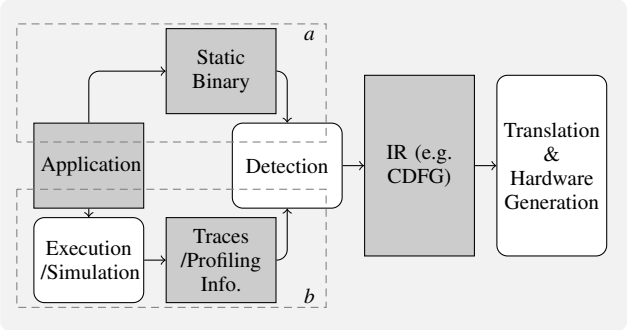

Fig. 1. Binary translation flow into custom hardware for *a* flows based on complied binary and *b* flows based on binary traces

After detection, segments are converted into *Control and Dataflow Graph* representations which expose the underlying *Instruction Level Parallelism* which we aim to exploit via automated hardware generation. Additionally, this lays the groundwork for memory access analysis which we may exploit for co-generation of specialized memory architectures in order to maximize memory access parallelism.

On-going work is addressing the extraction of cyclical execution traces or static code blocks (i.e., loops), the generation of hardware modules which implement all these binary segment types, and the automated integration and generation of the final hardware system. Future work will augment the framework with support for the RISC-V ISA, since ecosystem surrounding this specific architecture allows for greater possibilities in generation of custom computing architectures.

References

[Bel05] Fabrice Bellard. QEMU, a Fast and Portable Dynamic Translator. In *USENIX Annual Technical Conference, FREENIX Track*, pages 41–46. USENIX, 2005.

[PFC17] N. M. C. Paulino, J. C. Ferreira, and J. M. P. Cardoso. Generation of customized accelerators for loop pipelining of binary instruction traces. *IEEE Trans. on VLSI Systems*, 25(1):21–34, Jan 2017.

[Xil17] Xilinx. Vivado High-Level Synthesis. https://www.xilinx.com/products/design-tools/vivado/integration/esl-design.html, 2017. Accessed: 05-06-2017.